Compiler Projec1

2017030519 홍유진

실행 환경은 Linux Ubuntu 이다.

<Tiny compiler modification>

1. 실행 방법

make 후

1_Scanner 의 tinycompiler 디렉토리에 들어간후 컴파일

```
/1_Scanner/tinycompiler# ./tiny test.cm
```

2. 코드 설명

>>globals. H

```
#define MAXRESERVED 12

typedef enum
    /* book-keeping tokens */
   {ENDFILE,ERROR,
    /* reserved words */
    IF,ELSE, WHILE, RETURN, INT, VOID, THEN,END,REPEAT,UNTIL,READ,WRITE,
    /* multicharacter tokens */
    ID,NUM,
    /* special symbols */

    ASSIGN,EQ,LT,NE,LE,GT,GE,COMMA,LBRACE,RBRACE,LCURLY,RCURLY,PLUS,MINUS,TIMES,OVER,LPAREN,RPAREN,SEMI
   } TokenType;

extern FILE* source; /* source code text file */
```

새로운 symbol while, return, int, void, '==', '>=', '<=', ',' , '{' , '}' , '[' , ']' , '/**/'에 대한 enum 을 정의해줬다.
그리고 MAXRESERVED 값도 12 로 변경해줬다.

>> main.c

```
/* set NO_PARSE to TRUE to get a scanner-only compiler */
#define NO_PARSE TRUE
/* allocate and set tracing flags */
int EchoSource = TRUE;
int TraceScan = TRUE;
int TraceParse = FALSE;
```

Scanner 를 제작할 수 있도록 변수값을 변경해주었다.

>> Util.c

```
case IF:
case THEN:
case ELSE:
case END:
case REPEAT:
case UNTIL:
case READ:
case WRITE:
case WHILE:
case RETURN:
case INT:
case VOID:
  fprintf(listing,
     "reserved word: %s\n",tokenString);
  break;
```

```
case ASSIGN: fprintf(listing,"=\n"); break;
case LT: fprintf(listing,"<\n"); break;
case EQ: fprintf(listing,"==\n"); break;
case NE: fprintf(listing,"!=\n"); break;
case LE: fprintf(listing,"<=\n"); break;
case GT: fprintf(listing,">\n"); break;
case GE: fprintf(listing,">=\n"); break;
case LBRACE: fprintf(listing,"[\n"); break;
case RBRACE: fprintf(listing,"]\n"); break;
case LCURLY: fprintf(listing,"{\n"); break;
case RCURLY: fprintf(listing,"}\n"); break;
case COMMA: fprintf(listing,",\n"); break;
case LPAREN: fprintf(listing,"(\n"); break;
case RPAREN: fprintf(listing,")\n"); break;
case SEMI: fprintf(listing,";\n"); break;
case PLUS: fprintf(listing,"+\n"); break;
case MINUS: fprintf(listing,"-\n"); break;
case TIMES: fprintf(listing,"*\n"); break;
case OVER: fprintf(listing,"/\n"); break;
case ENDFILE: fprintf(listing,"EOF\n"); break;
```

Reserved word 에 추가된 case 와 새로 추가된 symbol 들을 출력할 수 있도록 했다.

>>Scan.c

```
   { START,NOT,INEQ, INCOMMENT, INNUM,INID,DONE,INLT,INGT,INNE,INOVER,INCOMMENT_}
/* state in scanner DFA */
typedef enum
```

: state 를 나타내줄 enum 을 추가하였다. INEQ 는 ==을 INNE 는 !=을 INCOMMENT_는 */을 INCOMMENT 는 /*를 INLT 와 INGT 는 >= <=를 알기 위해 추가되었다.'

```
/* lookup table of reserved words */
static struct
    { char* str;
      TokenType tok;
    } reservedWords[MAXRESERVED]
   = {{"if",IF},{"else",ELSE},{"while",WHILE},{"return",RETURN},{"int",INT},{"void",VOID},{"then",THEN},{"end",END},
     {"repeat",REPEAT},{"until",UNTIL},{"read",READ},
     {"write",WRITE}};
```

: reservedword 를 추가해주었다.

```
switch (state)
{ case START:
    if (isdigit(c))
      state = INNUM;
    else if (isalpha(c))
      state = INID;
    else if (c == '='){
          state = INEQ;
    }
    else if ((c == ' ') || (c == '\t') || (c == '\n'))
      save = FALSE;
   else if (c== '>'){
          state = INGT;
    }
    else if (c== '<'){
          state = INLT;
    }
    else if(c=='/'){
          save = FALSE;
          state = INOVER;
    }else if(c=='!'){
          state = INNE;
    }
```

: 각 토큰들이 다른 토큰이 될 가능성이 있는 토큰들의 상태를 정의해주었다. INEQ 는 = 이거나 ==, INGT 는 > 이거나 >=, INLT 는 < 이거나 <=, /는 나누기/이거나 /**/ !은 !=이 될 가능성이 있다. 그래서 다음 토큰또한 보고 state 를 결정하기 때문에 state 를 우선 지정해주었다.

```
      break;
    case '=':
      currentToken = EQ;
      break;
    case '<':
      currentToken = LT;
      break;
    case '+':
      currentToken = PLUS;
      break;
    case '-':
      currentToken = MINUS;
      break;
    case '*':
      currentToken = TIMES;
      break;
    case '(':
      currentToken = LPAREN;
      break;
    case ')':
      currentToken = RPAREN;
      break;
```

```
case '{':
  currentToken = LCURLY;
  break;
case '}':
  currentToken = RCURLY;
  break;
case '[':
  currentToken = LBRACE;
  break;
case ']':
  currentToken = RBRACE;
  break;
    case ';':
      currentToken = SEMI;
      break;
    case ',':
      currentToken=COMMA;
      break;
    default:
```

: 토큰 하나로 state 가 바로 결정될 수 있는 것들은 state 를 바로 지정해주었다.

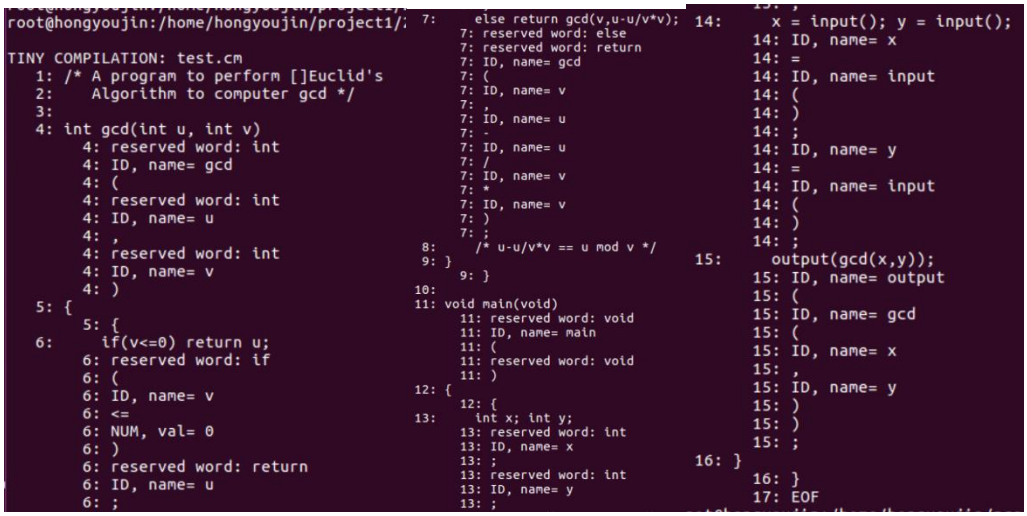
```
case INNE:
    state = DONE;
    if(c=='=')
          currentToken=NE;
    else{
        currentToken = ERROR;
    }
    break;
case INCOMMENT:
    save = FALSE;
    if (c == EOF)
    { state = DONE;
      currentToken = ENDFILE;
    }
    else if (c == '*') state = INCOMMENT_;
    break;
case INLT:
    state = DONE;
    if(c=='=')
          currentToken = LE;
    else{
          ungetNextChar();
          currentToken = LT;
    }
    break;
case INOVER:
    if(c=='*'){
          state = INCOMMENT;
          save = FALSE;
    }else{
          ungetNextChar();
          state = DONE;
          currentToken=OVER;
    }
    break;
```

```
case INCOMMENT_:
    save = FALSE;
    if(c==EOF){
          state = DONE;
          currentToken=ENDFILE;
    }else if (c=='/') state = START;
    else state = INCOMMENT;
   break;
case INGT:
    state = DONE;
    if(c=='=')
          currentToken= GE;
    else{
          ungetNextChar();
          currentToken = GT;
    }
    break;
case INEQ:
    state = DONE;
    if(c=='=')
          currentToken = EQ;
    else
    {
          currentToken = ASSIGN;
          ungetNextChar();
    }
    break;
```

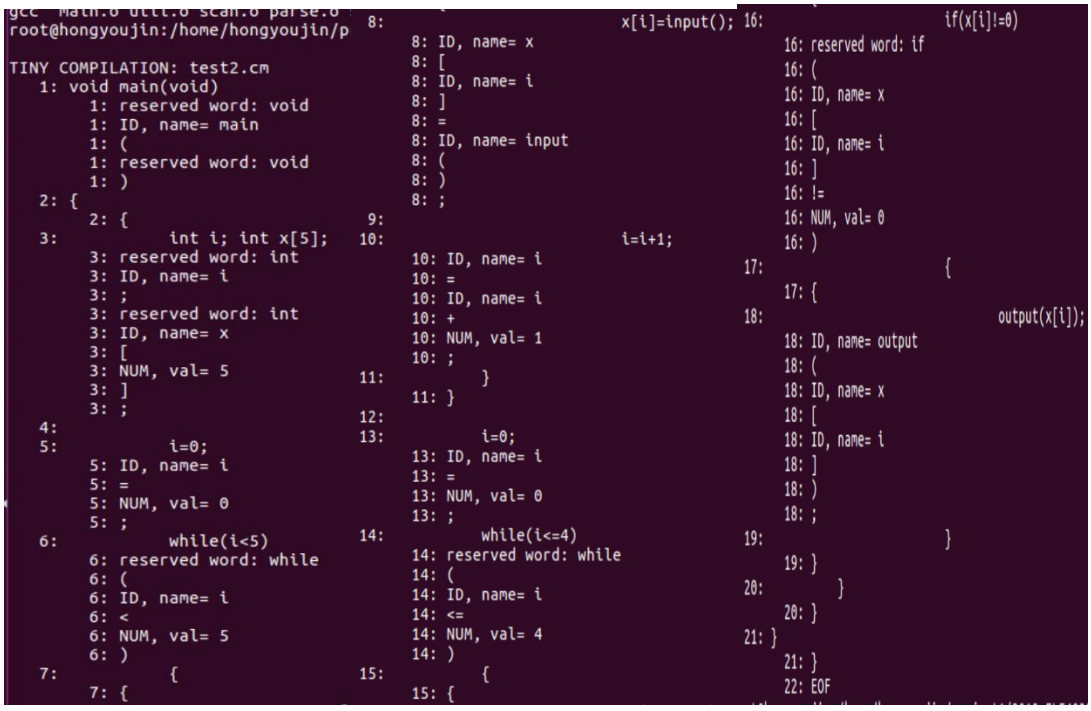
: 위의 정의한 state 들이 정확히 어떤 state 인지 판단할 수 있는 코드를 작성하였다.
/**/는 처음 /를 보고 INOVER 로 인식 후 *이 있으면 INCOMMENT 로 state 를 변경한다. State 가 INCOMMENT 이고 *이 있으면 바깥쪽 주석처리하는 부분이므로 INCOMMENT_로 state 를 변경한다. 그 후 state 가 INCOMMENT_이면 /가 나오면 주석이 끝났다는 표현이므로 state 를 START 로 변경시킨다.

3. 실행 결과

- test.cm



-test2.cm



<lex compiler modification>

1.실행 방법

Makefile 을 수정한다.

```
OBJS = main.o util.o scan.o parse.o symtab.o analyze.o code.o cgen.o
OBJS_FLEX = main.o util.o lex.yy.o parse.o symtab.o analyze.o code.o cgen.o

.PHONY:all scanner_cimpl scanner_flex $(OBJS) $(OBJS_FLEX) lex.yy.c

all: scanner_cimpl scanner_flex

scanner_cimpl:$(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o $@

scanner_flex:$(OBJS_FLEX)
    $(CC) $(CFLAGS) $(OBJS_FLEX) -o $@ -lfl

lex.yy.o:cminus.l
    flex $^
    $(CC) $(CFLAGS) -c lex.yy.c
```

Make 후 1_Scanner/lexcompiler 디렉토리로 들어간 후

```
./scanner_flex test.cm
```

2.코드 설명

>> Globals.h util.h

: tiny compiler 와동일하게 수정한다.

>>Cminus.l

```
"if"            {return IF;}
"then"          {return THEN;}
"else"          {return ELSE;}
"end"           {return END;}
"repeat"        {return REPEAT;}
"until"         {return UNTIL;}
"read"          {return READ;}
"write"         {return WRITE;}
"int"           {return INT;}
"while"         {return WHILE;}
"void"          {return VOID;}
"["             {return LBRACE;}
"]"             {return RBRACE;}
"return"        {return RETURN;}
"="            {return ASSIGN;}
"=="            {return EQ;}
"<"             {return LT;}
">"             {return GT;}
"<="            {return LE;}
">="            {return GE;}
"!="            {return NE;}
"+"             {return PLUS;}
"-"             {return MINUS;}
"*"             {return TIMES;}
"/"             {return OVER;}
"("             {return LPAREN;}
")"             {return RPAREN;}
";"             {return SEMI;}
{number}        {return NUM;}
{identifier}    {return ID;}
{newline}       {lineno++;}
{whitespace}    {/* skip whitespace */}
"{"             {return LCURLY;}
"}"             {return RCURLY;}
","             {return COMMA;}
```

```
"/*"            { char c, tmp=NULL;
                  do{
                        c = input();
                        if(c==EOF) break;
                        if (c=='\n') lineno++;
                        if(tmp=='*'&&c=='/' ) break;
                        tmp = c;
                  }while(c);
                }
                { return ERROR;}
```

새로운 symbol 들을 return 해준다. 이때 /**/만 do-while 문을 이용하여 /**/을 판단할 수 있도록 구현하였다.

3.실행 결과

<test.cm>

<test2.cm>